Detailed Diagnosis of Performance Anomalies in Sensornets

Tony O'Donovan[‡], Nicolas Tsiftes[♯], Zhitao He[♯], Thiemo Voigt[♯], and Cormac J. Sreenan[‡]
[‡]University College Cork
[♯]Swedish Institute of Computer Science

Abstract

We address the problem of analysing performance anomalies in sensor networks. In this paper, we propose an approach that uses the local flash storage of the motes for logging system data, in combination with online statistical analysis. Our results show not only that this is a feasible method but that the overhead is significantly lower than that of communication-centric methods, and that interesting patterns can be revealed when calculating the correlation of large data sets of separate event types.

1 Introduction

Deployments of sensor networks can experience surprisingly poor results when challenged by unexpected conditions. Possible causes of failures include faulty software [10], challenging weather conditions [1], and complex interactions between protocols [4]. Understanding the exact causes of failures and poor performance is difficult, since it may be impossible to provide detailed data reports using the radio in such a situation.

In this paper, we tackle the problem of analysing performance problems in deployed sensor networks. Our objective is to be able to analyse log data of high fidelity—even if the communication exhibits severe problems. Earlier methods for analyzing performance problems typically depend on having a communication channel to each mote in the network, through which live debug data is requested from a sink [14]. Another method is to have extra motes listening on radio traffic and reporting problems [2, 16]. This method requires additional hardware to be deployed, however, and can only deduce information from the overheard traffic.

We exploit the storage capabilities of motes to enable both *in-situ* and *post-mortem* performance analysis of sensor networks. Each mote in a network periodically stores detailed performance data, which can either be analysed locally or collected later for analysis on a PC. Our in-situ analysis software includes *Pearson's correlation* and *mutual information* to demonstrate that interesting results can be obtained by the motes themselves. Our method is not limited to these formulas, however: since we can obtain all data for post-mortem analysis, more comprehensive statistical analyses can be employed if necessary.

The contributions of this paper are the design and evaluation of a storage-centric scheme for the detailed diagnosis of performance anomalies in sensornets. We demonstrate that we can store large quantities of data efficiently, and show the node-local analysis possible on the stored data. The performance information can be stored at much higher fidelity than is practical with a communication-centric approach, also improving post-mortem analysis capabilities. Our results show that the energy consumption of transmitting and receiving four packets, in optimal conditions and over one hop, is similar to reading 2,000 logged samples from the on-node file system and computing a correlation function.

2 Motivation

Anecdotal evidence of sensor network deployments clearly indicates that they are error-prone and often perform poorly [10, 17]. Environmental conditions, component failure, and programming error can all cause problems. Hence, a variety of debugging tools have been developed to help researchers to understand faults. Such tools commonly require that state be sent to a sink, either at regular intervals or upon a query from a network operator [14].

In recent work, Gnawali et al. emphasise that, from a system implementation point of view, providing a detailed logging layer was the most important design decision in CTP Noe [9]—a protocol that has been tested and debugged in a large variety of environments. All of these environments, however, are in-house testbeds where the events can be reported over a serial line. In contrast, real deployments are limited to delivering debug data over radio.

The problem with mixing debug packets and ordinary packets in the network is that 1) the packets may not get through in case of bad performance, 2) "heisenbugs" may be introduced, and 3) it is limited to low-fidelity data. Although frequent sampling and reporting gives a more accurate picture of the network, it requires extra traffic. In large networks, this cost can be considerable. Given a fixed and often tight energy budget for fault diagnosis, only a limited

number of variables can be reported to the sink regularly, and only at moderate rates.

By logging sensor values together with performance statistics, we enable correlation of arbitrary data sets, self-monitoring, and detailed diagnosis of performance anomalies. Earlier research has uncovered unexpected correlations between performance and different types of environmental characteristics, or hardware failures. Boano et al. [1] have shown that the temperature has a significant effect on the packet reception rate (PRR), even inside an office building. Finne and Eriksson observed that radio communication triggered sensor readings, causing superfluous alarms [7].

3 Design Overview

There are three distinct elements to the storage-centric approach. Firstly, the storage itself, which relates to what data is stored, how frequently, and for how long it should be kept. Secondly, the computational capabilities of the motes are employed to perform some statistical analysis, allowing the nodes to detect and possibly even diagnose performance problems. Finally, data collection involves the methods available to the network operator for accessing the stored data, whether it is only a summary, a subset, or all of the information that is required.

3.1 Storage

A simple, yet essential point of our work is to exploit the node-local storage that exists in many types of motes. Unlike traditional debugging tools that mainly support queries for live data only, our approach supports analysis of detailed data collected over a long time. For this purpose, we use the Coffee file system [19] in Contiki to store vast amounts of sensor data and networking statistics in a circular log. Coffee is able to append data in files at a speed close to that of the underlying flash driver, and uses a constant RAM footprint for each file, regardless of size.

The online logs accommodate arbitrary metrics that are relevant to the performance. Each mote is able to scan and analyse its own performance, sending only statistical summaries or warning messages back to the sink. After inspecting these summaries the network operator can decide to download more detailed data from a node that reports anomalous performance. The implication is that analysis of the network performance is significantly less likely to affect the behavior of the network. Moreover, even without having some form of analysis online, the stored data is useful in post-deployment analysis of performance, effectively removing the need for any debug-data messages to be sent.

3.2 Statistical Analysis

Diagnosis of a performance problem requires analysis of the data available to determine the cause. Given the information stored on the motes and their computational capabilities, many techniques are possible. Examples include finding the maximum, minimum, or average of a variable–such as RSSI–over a specified period, or searching for deviations in the logs. Another option is to examine different sets of events looking for any correlation between them.

In order to illustrate the type of online analysis possible, we have implemented two functions on the motes for calculating such correlations using standard statistical methods. The first function is Pearson's correlation coefficient r_{xy}, between the sets x and y as shown in Equation 1.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \quad (1)$$

In this equation, x_i and y_i are the i-th measurements of two series of n measurements of the random variables X and Y. $\bar{x}$ and $\bar{y}$ denote the sample means of X and Y. Using a medium effect size of 0.3 and alpha error 0.05, an estimated 84 records or more are needed to detect a significant correlation (with 80% power).

Our implementation of this function retrieves the data series from the Coffee file system and computes the correlation. We implement Pearson's correlation coefficient as an iterative algorithm that reads one sample and computes one step. By only storing a small number of intermediate data samples, we are able to keep the memory footprint of this function low. We need to make two passes over all samples: the first one to compute the average and the second for the sums in the equations.

Pearson's correlation coefficient is widely used but cannot compute the correlation between a binary and a continuous variable. Therefore, we have also implemented an algorithm to compute the mutual information $I(X;Y)$ between a discrete variable X and a continuous random variable Y. $I(X;Y)$ is also a more generic correlation measure than Pearson's correlation coefficient.

$$I(X;Y) = \frac{1}{2}\log(2\pi e\sigma^2) - \frac{1}{2}\sum_{x\in X} P(x)\log(2\pi e\sigma_x^2) \quad (2)$$

We compute mutual information as depicted in Equation 2. In this equation, σ^2 is the variance of the continuous variable Y and σ_x^2 is the variance of Y given that $X = x$. The implementation of the algorithm is similar to the one for Pearson's coefficient described above in that we need to make two passes over the data. The first to compute the averages and the second to compute the variances.

3.3 Data Collection

Performance anomalies have many causes, as already outlined, that can result in a variety of problems. For example, congestion may result in increased delivery delays and intermittent connectivity, whereas a node failure will result in a disconnection from the affected node and possibly any nodes that use it to forward messages. As a result several methods for accessing the stored data are required.

When there is no physical access to a deployed network, the preferred method is remote querying. The network operator is able to request and receive data summaries from selected nodes in the network. Upon receipt of such a query, the node performs the requested calculation, returning the results required. The correlation procedures described in Section 3.2 are examples of this. Since only summaries are sent, this method has little impact on the network.

Alternatively the operator can take a mobile sink (e.g, a laptop) into the field to query nodes directly. For partitioned networks this may be the only option available. This method has a smaller response delay and a lower overhead since all

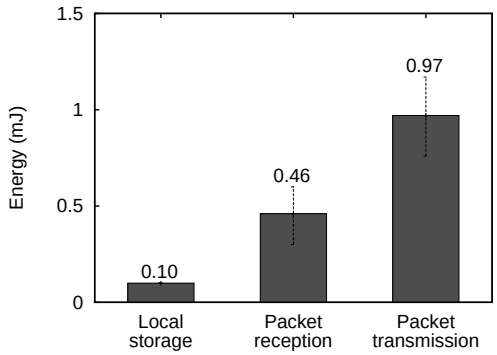

Figure 1: Storing performance data to the flash requires significantly less energy than sending the data over the radio, even under optimal radio conditions.

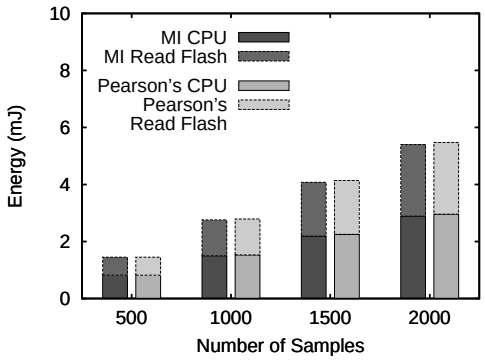

Figure 2: The energy consumption for computing a Pearson's correlation coefficient and mutual information (MI) from locally stored data.

communication is single-hop, making it suitable for diagnosis isolated to a small section of the network.

To allow for deeper analysis than what is practical on the nodes, all stored data can be downloaded in a batch transfer over the radio or using a node's UART bus. While a UART download has no impact on network operation, it does require a physical connection to the node. A batch download over the radio is best done using a separate channel or a mobile sink to avoid forwarding the data over the network.

4 Evaluation

To evaluate our storage-centric approach to performance debugging, we quantify the energy required to store information locally on a node in comparison to sending it via the radio. We also use the motes to efficiently calculate statistical correlation among events. Finally, we collect data from a testbed to demonstrate typical observations that are possible using storage-centric performance analysis.

4.1 Energy Analysis

In this experiment, we compare the cost of logging data locally with that of transmitting it over the radio. Our setup consists of a set of emulated Tmote Sky motes in the cycle-accurate Cooja/MSPsim simulator [6]. Each log record is 64 bytes, comprising debug information from several parts of the system.

We run the Contiki operating system on the motes, and use the Coffee file system and the unicast primitive in Rime [5]. The MAC protocol under Rime is X-MAC, configured with a 1.25 % duty cycle. The duty cycle is selected using the default radio on-time of $\frac{1}{160}$ s, and an off-time of 0.5 s, which is half of the average packet transmission interval. We send 1,000 packets and measure the per-packet energy consumption using Contiki's Compower library.

Figure 1 shows that the energy consumption is considerably lower for storing the data locally. Whereas the packet costs vary, the cost of storing the data locally is the same since Coffee's optimised file append operation closely follows the performance of the underlying flash device driver.

We have used an optimal setup for unicast transmissions: a single-hop network without interference. The total cost of transmitting the data from a mote to a sink in a multi-hop network is a multiple of the single-hop cost. Furthermore, the contention and the collision rate in the network will increase significantly if performance debugging packets are transmitted at high rates: our 1,000 packets sent in the network layer generated 3,762 packets in the link layer because of the strobing procedure in X-MAC.

Using Contiki's energy estimation utility, we also characterize the energy overhead of logging over a range of sample sizes [50, 200]. The average overhead is 0.435 μ J per stored byte, with maximum and average estimation errors of 5.3% and 2.1% respectively.

4.2 Computing Correlations

The purpose of this experiment is to determine if any correlation exists between the environmental conditions and the minimum transmission power required for successful communication (TX_{min}). We use a data set that we obtained previously from an office testbed [1]. The data consists of TX_{min} along with samples collected from the nodes' temperature, humidity, and light sensors. Tmote Sky nodes are equipped with two light sensors; a photo-active radiation sensor (PAR) and an ambient light sensor (TSR). The data is measured over 48 hours, during which time-significant fluctuations in the data were observed.

To prepare the calculation, we transfer the data to a mote and store it in a file. We compute the correlation between TX_{min} and the temperature using Pearson's correlation coefficient using Equation 1, then we compute the mutual information (MI) using Equation 2. In contrast to Pearson's correlation coefficient, MI allows us to compute the correlation between a binary variable and a continuous one.

We use Contiki's power profiler to measure the energy consumption for reading the data from the file system and computing the correlations. We vary the amount of samples used as input to the correlation functions. We have also verified that the computed correlation is correct.

4.2.1 Pearson's Correlation

Figure 2 confirms our expectation that the energy consumption increases linearly with the sample size where one sample consists of a TX_{min} and a temperature value. The figure also shows the power consumption for computing the Pearson's correlation coefficient is low. In particular, the cost

Table 1: The calculated Pearson's Correlation Coefficient.

Sensor	Sender	Receiver
Temperature	0.779	0.670
Humidity	-0.776	-0.689
TSR	0.641	0.608
PAR	0.640	0.590

of reading the file from flash is lower than the cost of the CPU activity.

The results of the Pearson's correlation coefficient calculation are shown in Table 1. This value is between 1 and -1, with values approaching ± 1 indicating that there is correlation between the data sets. Values close to 0 denote that no correlation exists. The results show that TX_{min} correlates with the temperature, consistent with the findings by Boano et al. [1]. Incidentally for this deployment, we observe that TX_{min} also correlates with humidity and light.

4.2.2 Mutual Information

We use a modified version of the trace in Section 4.2.1 to calculate the mutual information. In the trace, we pick a transmission power TX and convert TX_{min} into a binary variable by setting it to 0 if $TX_{min} > TX$ and 1 otherwise. As in the previous section we compute the energy consumption. Figure 2 shows that the results are very similar. The reason is that most of the CPU power is used for reading the files and computing the sums. The calculation of mutual information uses slightly less energy because the logarithm is cheaper to compute than the square root.

Reading 2000 samples from a file and applying one of the two correlation functions takes less than 300 ms, and can hence be scheduled without affecting periodic tasks such as sensing and logging. If needed, there is also the option of yielding control of the processor at multiple points within the iterative algorithms.

4.2.3 Packet Transmission vs. Local Analysis

When comparing the results in Figure 1 and Figure 2, we see that reading 500 samples from flash and computing a correlation requires the same amount of energy as transmitting and receiving one packet over a single hop. The energy consumption of transmitting and receiving four packets, in optimal conditions, is similar to reading 2,000 samples and computing the correlation. These results clearly demonstrate the applicability of our approach.

4.3 Case Study of a Convergecast Application

We run a convergecast application on our 17-node TelosB testbed. Each node takes a sample every 10 seconds. A sample consists of the values of 4 sensors, 6 energy estimates, and 18 Rime statistics variables. The log module stores all the current samples in a 108-byte record together with a sample ID number and a time stamp. Every minute, the node constructs a message from the latest sensor and energy estimates, and sends it to the sink using the Rime collect protocol over X-MAC; the sink augments the received message with its own time stamp, and stores it in a file.

We collect over 8,000 samples in each node during a 24-hour experiment, with more than 1,500 messages per node sent to the sink. Some of these messages are lost after per-hop retransmission fails repeatedly and a new route is not found in due time. Using the information at the sink alone, we calculate the PRR for each node. By observing gaps in the message sequence numbers, we can also detect fluctuations in the connectivity.

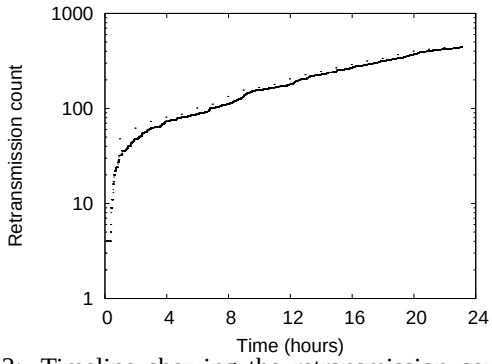

Figure 3: Timeline showing the retransmission count for node 106 from data collected after deployment.

Table 2: Summary of node connectivity observed by sink

Node ID	PRR (%)	Tx/Msg	Hops/Msg
17	49	1.77	1.67
59	58	1.93	1.86
103	67	2.13	2.08
106	15	4.91	4.57

We summarise the sink log in Table 2. It is difficult to reason about the causes of packet losses using only the limited log at the sink. The local logs provide extra information including the Rime statistics, reflecting changes at a finer time resolution and giving a clearer picture of network operation. Figure 3 is an example of this, showing the retransmission count node 106 logged over the 24-hours.

After inspecting the detailed logs stored in the motes, we discovered more unexpected problems. A power failure at about 4 AM stopped several nodes. Later we were able to recover all the local logs. By inspecting the time stamps of the last record, we find the exact moment of the event. We also found that among the 8,000+ time stamps, a few consecutive time stamps were wrong by one hour. This bug is the cause of the outliers in Figure 3 and is due to a peculiar flaw in our calendar time library. We would find it difficult to analyse this bug using only the coarse log of the sink.

5 Related Work

Sympathy is a network-monitoring tool for periodic data gathering deployments [14]. The sink collects and analyses performance metrics from all nodes in the network. Unlike Sympathy, PAD and PerDB employ a passive approach to detection by embedding a small amount of performance-related data in application messages [11, 13]. PAD uses belief networks and causal diagrams for anomaly diagnosis, similar to Sympathy's decision tree. The Visibility metric is obtained by adding weights to a decision tree's branches to aid in the

design of network protocols that are easier to debug [20].

Using changes in routing and traffic patterns is a common method for fault detection, as seen in Sympathy, PAD and PerDB. Our storage-centric approach allows fault detection on the nodes themselves, however, not the sink. Like PerDB and PAD, one of our goals is to reduce the overhead of performance debugging, but while those systems discard data not embedded in application messages, we log it on the nodes' external flash for use in detection and diagnosis.

EnviroLog and our system both allow specified data to be stored on a node's flash memory for the purpose of improving performance [12]. The authors' main goal is to enable an exact replay of recorded events and conditions to statistically analyse network performance. Our aim is to improve detection and analysis of performance problems using the nodes' storage and computational capabilities.

Clairvoyant uses RPC to give the network operator some debugging capabilities [21]. It provides a fully functional debugger, but can affect code execution and also suffers from increased network traffic. PD2 uses debugging to localise performance anomalies post-deployment [3]. While our system doesn't provide debugging functionality, it does allow recording of data for analysis, at little cost.

Tavakoli et al. provide global state monitoring and debugging using a predicate specification, aiming to enable easier troubleshooting of real-world networks [18]. We choose instead to store debug data on the nodes themselves giving significant energy saving over sending it via the radio.

Suelo analyses sensor data looking for predefined features that indicate sensor faults [15]. Ganeriwal et al. use outlier detection to find invalid data from faulty or compromised nodes [8]. Instead of using statistical analysis for application-layer data integrity, as these systems do, we use such analysis to find anomalies in the operation of the communication layers.

6 Conclusions

We present a storage-centric approach for analysing performance anomalies in deployed sensor networks. We evaluate the system, demonstrating not only is it feasible to store and analyse large quantities of data on the nodes, but also that it is more energy efficient than sending the information to the sink. In addition we show that with the large quantities of data this scheme can log it is possible to observe the network operation in great detail, making it easier to diagnose performance problems.

Acknowledgments

Thanks to Daniel Gillblad (SICS) for insights on algorithms for computing correlations. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 224282. This work has been partially supported by CONET, the Cooperating Objects Network of Excellence.

7 References

[1] C. A. Boano, N. Tsiftes, T. Voigt, J. Brown, and U. Roedig. The Impact of Temperature on Outdoor Industrial Sensornet Applications. In *IEEE Transactions on Industrial Informatics*, accepted for publication.

[2] B. Chen, G. Peterson, G. Mainland, and M. Welsh. Livenet: Using passive monitoring to reconstruct sensor network dynamics. In *Proceedings of Distributed Computing in Sensor Systems (DCOSS)*, 2008.

[3] Z. Chen and K. Shin. Post-deployment performance debugging in wireless sensor networks. In *Proceedings of the Real-Time Systems Symposium (IEEE RTSS)*, 2009.

[4] J. Choi, M. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In *Proceedings of ACM SenSys*, 2009.

[5] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of ACM SenSys*, 2007.

[6] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, and T. Voigt. Accurate Network-Scale Power Profiling for Sensor Network Simulators. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2009.

[7] N. Finne, J. Eriksson, A. Dunkels, and T. Voigt. Experiences from two sensor network deployments self-monitoring and self-configuration keys to success. In *Proceedings of WWIC*, 2008.

[8] S. Ganeriwal, L. Balzano, and M. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 2008.

[9] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of ACM SenSys*, 2009.

[10] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[11] K. Liu, M. Li, Y. Liu, M. Li, Z. Guo, and F. Hong. Passive diagnosis for wireless sensor networks. In *Proceedings of ACM SenSys*, 2008.

[12] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2006.

[13] V. Pejovic and C. Sreenan. PerDB: Performance Debugging for Wireless Sensor Networks. In *European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, 2009.

[14] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of ACM SenSys*, 2005.

[15] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, and D. Estrin. Suelo: human-assisted sensing for exploratory soil monitoring studies. In *Proceedings of ACM SenSys*, 2009.

[16] M. Ringwald, K. Römer, and A. Vitaletti. Passive inspection of sensor networks. In *Proceedings of Distributed Computing in Sensor Systems (DCOSS)*, 2007.

[17] T. Schmid, H. D. Ferrière, and M. Vetterli. Sensorscope: Experiences with a wireless building monitoring sensor network. In *Organization of the Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2005.

[18] A. Tavakoli, D. Culler, and S. Shenker. The case for predicate-oriented debugging of sensornets. In *Proceedings of the Workshop on Hot Topics in Embedded Networked Sensor Systems (HotEmnets)*, 2008.

[19] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling large-scale storage in sensor networks with the coffee file system. In *Proceedings of ACM/IEEE IPSN*, 2009.

[20] M. Wachs, J. Choi, J. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis. Visibility: A new metric for protocol design. In *Proceedings of ACM SenSys*, 2007.

[21] J. Yang, M. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. *Proceedings of ACM SenSys*, 2007.